Clean Copy

Patent Application of

Kevin W Jameson

For

COLLECTION CONTENT CLASSIFIER

CROSS REFERENCES TO RELATED APPLICATIONS

The present invention uses inventions from the following patent applications, that are filed contemporaneously herewith, and which are incorporated herein by reference:

USPTO 09/885078, Collection Information Manager; Kevin Jameson.

FIELD OF THE INVENTION

This invention relates to automated software systems for processing collections of computer files in arbitrary ways, thereby improving the productivity of software developers, web media developers, and other humans and computer systems that work with collections of computer files.

Prior Art Makefile Generators

Makefile generator programs generate makefiles for humans who are building software programs. Typically, makefiles contain computer instructions for compiling source code files and linking compiled object files to produce executable files or libraries of object files. In addition, programmers typically include a variety of other useful command sequences in makefiles to increase productivity.

Examples of popular freeware makefile generators include automake, imake, and mkmf (make makefile). Although each program is useful, each program has several important classification shortcomings.

Automake has no dynamic content discovery mechanism; instead it requires programmers to manually list all files that require processing. Neither does it have a mechanism for sharing classification information, so multiple automake files cannot easily share user-provided information. Finally, it uses an input file that must be manually constructed, and so its classification operations are not fully automated.

Imake has no dynamic content discovery mechanism; instead it requires programmers to manually list all files that require processing. It also uses an input file that must be manually constructed, so its classification operations are not fully automated.

Mkmf does have a dynamic content discovery mechanism that dynamically includes all source files in the current directory in the output makefile. However, only the current directory is used to find source files; no other directories are supported. Significantly, all source files in the directory are included, whether they should be included or not. This forces programmers to unnaturally restrict the directory contents to only those file that should be discovered by mkmf. Finally, all files are used to build one product only; files cannot be grouped into multiple products.

Thus these makefile generators, which are characteristic of the prior art, have significant

DETAILED DESCRIPTION

Overview of Collections

This section introduces collections and some related terminology.

Collections are sets of computer files that can be manipulated as a set, rather than as individual files. Collection information is comprised of three major parts: (1) a collection specifier that contains information about a collection instance, (2) a collection type definition that contains information about how to process all collections of a particular type, and (3) optional collection content in the form of arbitrary computer files that belong to a collection.

Collection specifiers contain information about a collection instance. For example, collection specifiers may define such things as the collection type, a text summary description of the collection, collection content members, derivable output products, collection processing information such as process parallelism limits, special collection processing steps, and program option overrides for programs that manipulate collections. Collection specifiers are typically implemented as simple key-value pairs in text files or database tables.

Collection type definitions are user-defined sets of attributes that can be shared among multiple collections. In practice, collection specifiers contain collection type indicators that reference detailed collection type definitions that are externally stored and shared among all collections of a particular type. Collection type definitions typically define such things as collection types, product types, file types, action types, administrative policy preferences, and other information that is useful to application programs for understanding and processing collections.

Collection content is the set of all files and directories that are members of the collection. By convention, all files and directories recursively located within an identified set of

subtrees are usually considered to be collection members. In addition, collection specifiers can contain collection content directives that add further files to the collection membership. Collection content is also called collection membership.

Collection is a term that refers to the union of a collection specifier and a set of collection content.

Collection information is a term that refers to the union of collection specifier information, collection type definition information, and collection content information.

Collection membership information describes collection content.

Collection information managers are software modules that obtain and organize collection information from collection information stores into information-rich collection data structures that are used by application programs.

Collection Physical Representations -- Main Embodiment

Figures 1-3 show the physical form of a simple collection, as would be seen on a personal computer filesystem.

FIG 1 shows an example prior art filesystem folder from a typical personal computer filesystem. The files and directories shown in this drawing do not implement a collection 100, because no collection specifier 102, FIG 2 Line 5 exists to associate a collection type definition FIG 4 101 with collection content information FIG 4 103.

FIG 2 shows the prior art folder of FIG 1, but with a portion of the folder converted into a collection 100 by the addition of a collection specifier file FIG 2 Line 5 named "cspec". In this example, the collection contents FIG 4 103 of collection 100 are defined by two implicit policies of a preferred implementation.

First is a policy to specify that the root directory of a collection is a directory that contains a collection specifier file. In this example, the root directory of a collection 100 is a directory named "c-myhomepage" FIG 2 Line 4, which in turn contains a collection specifier file 102 named "cspec" FIG 2 Line 5.

Second is a policy to specify that all files and directories in and below the root directory of a collection are part of the collection content. Therefore directory "s" FIG 2 Line 6, file "homepage.html" FIG 2 Line 7, and file "myphoto.jpg" FIG 2 Line 8 are part of collection content FIG 4 103 for said collection 100.

FIG 3 shows an example physical representation of a collection specifier file 102, FIG 2 Line 5, such as would be used on a typical personal computer filesystem.

Collection Information Types

Figures 4-5 show three kinds of information that comprise collection information.

FIG 4 shows a high-level logical structure of three types of information that comprise collection information: collection processing information 101, collection specifier information 102, and collection content information 103. A logical collection 100 is comprised of a collection specifier 102 and collection content 103 together. This diagram best illustrates the logical collection information relationships that exist within a preferred filesystem implementation of collections.

FIG 5 shows a more detailed logical structure of the same three types of information shown in FIG 4. Collection type definition information FIG 4 101 has been labeled as per-type information in FIG 5 103 because there is only one instance of collection type information 101 per collection type. Collection content information FIG 4 103 has been labeled as per-instance information in FIG 5 103 because there is only one instance of collection content information per collection instance. Collection specifier information 102 has been partitioned into collection instance processing information 104, collection-

type link information 105, and collection content link information 106. FIG 5 is intended to show several important types of information 104-106 that are contained within collection specifiers 102.

Suppose that an application program means FIG 6 110 knows (a) how to obtain collection processing information 101, (b) how to obtain collection content information 103, and (c) how to relate the two with per-collection-instance information 102. It follows that application program means FIG 6 110 would have sufficient knowledge to use collection processing information 101 to process said collection content 103 in useful ways.

Collection specifiers 102 are useful because they enable all per-instance, non-collection-content information to be stored in one physical location. Collection content 103 is not included in collection specifiers because collection content 103 is often large and dispersed among many files.

All per-collection-instance information, including both collection specifier 102 and collection content 103, can be grouped into a single logical collection 100 for illustrative purposes.

Collection Application Architectures

Figures 6-7 show example collection-enabled application program architectures.

FIG 6 shows how a collection information manager means 111 acts as an interface between an application program means 110 and collection information means 107 that includes collection information sources 101-103. Collectively, collection information sources 101-103 are called a collection information means 107. A collection information manager means 111 represents the union of all communication mechanisms used directly or indirectly by an application program means 110 to interact with collection information sources 101-103.